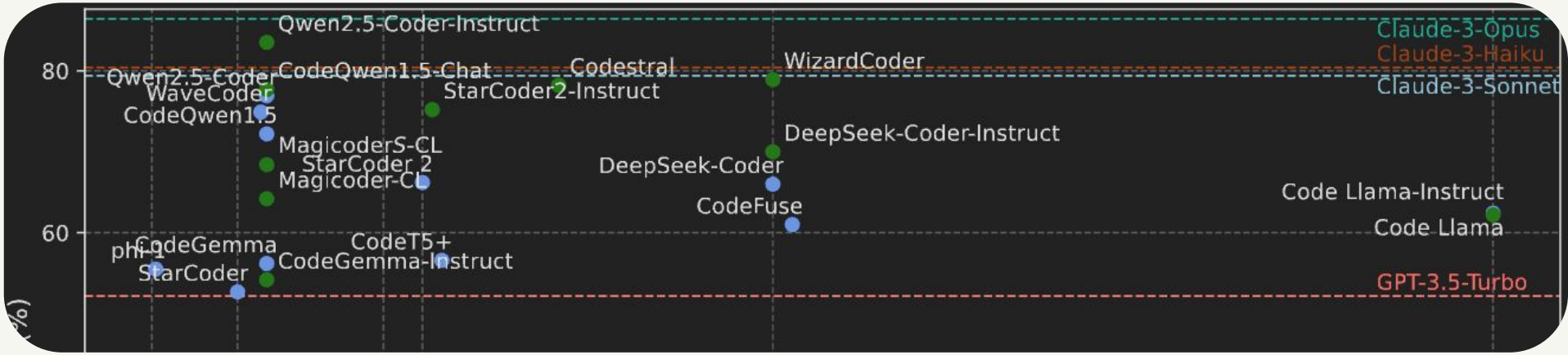


A Survey on Large Language Models for Code Generation

CS 5914

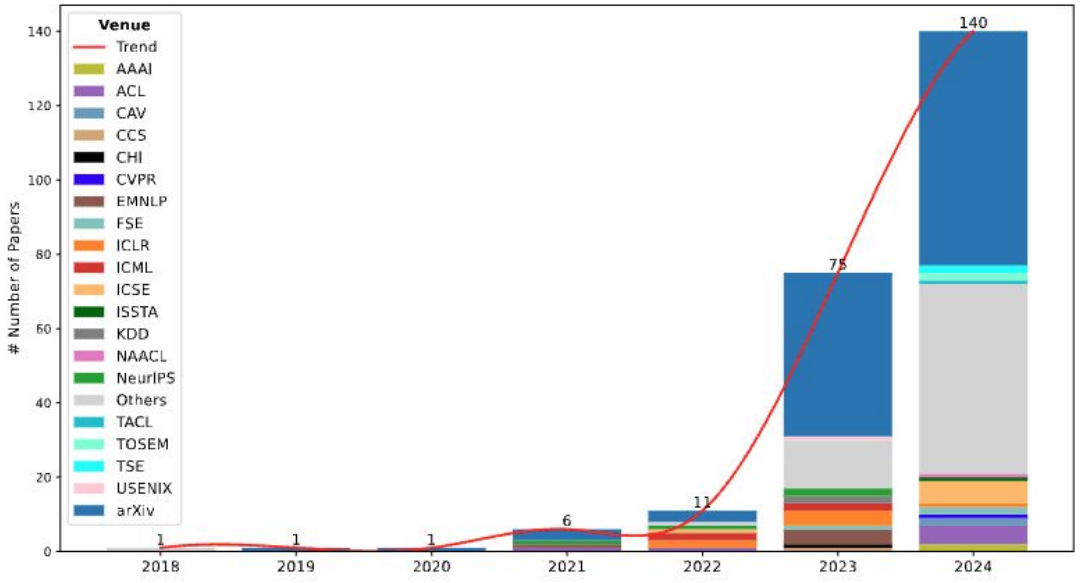
JUYONG JIANG*, The Hong Kong University of Science and Technology (Guangzhou), China
FAN WANG*, The Hong Kong University of Science and Technology (Guangzhou), China
JIASI SHEN†, The Hong Kong University of Science and Technology, China
SUNGJU KIM†, NAVER Cloud, South Korea
SUNGHUN KIM†, The Hong Kong University of Science and Technology (Guangzhou), China

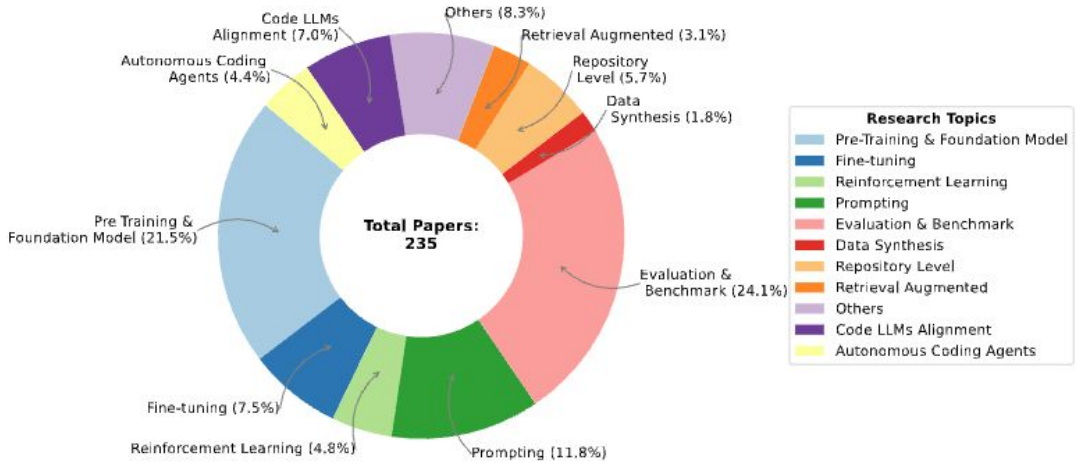
Presented by Prayash Joshi, Heesang Han

Feb, 11, 2025

Problem Statement

This survey aims to to "**bridge this gap** by providing a **systematic literature review** that serves as a valuable **reference** for researchers investigating the **cutting-edge progress in LLMs for code generation.**"

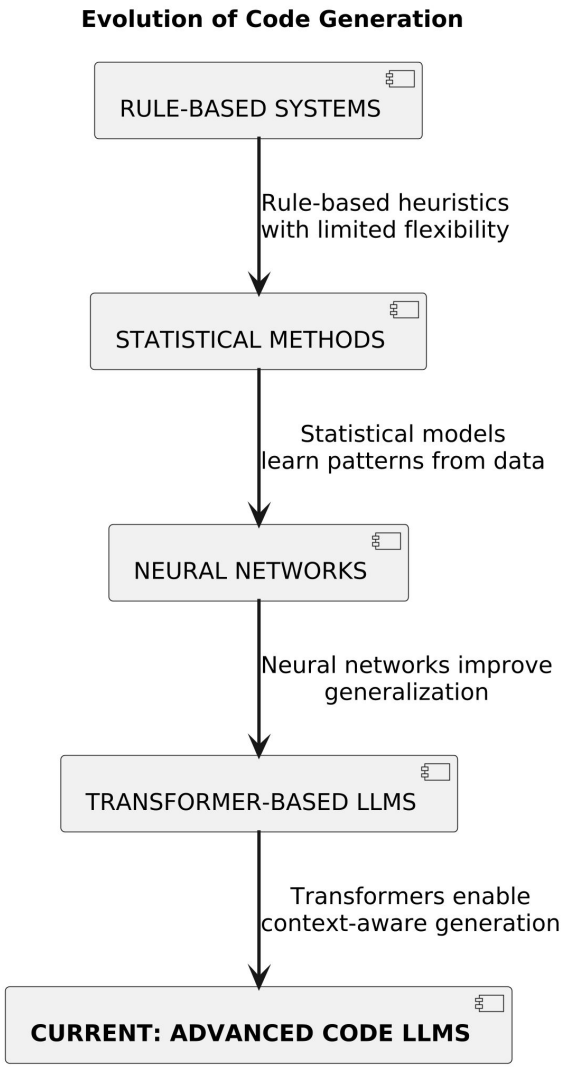




Motivation

Main motivations for studying Code Generation using LLMs:

1. **Paradigm Shift in Development**
   - rule-based to AI-driven generation
2. **Democratization & Productivity**
   - Reduced barrier to entry for coding
3. **Industry Transformation**
   - Growing impact on software development practices



Background

1. Model Architectures
    a. **Encoder-decoder** : Dual-purpose architecture for understanding and generating code
    b. **Decoder-only** : Specialized for generation tasks
2. Scale vs. Scope
    a. General Purpose vs. Code Instructed
        - Foundational Models (GPT-4, Claude)
        - Specialized for programming tasks (Code Llama, Starcoder)
    b. Training for Code Instructed
        - Pre-training on large code repositories
        - Instruction tuning for task alignment
    c. Reinforcement Learning Refinements
        - Feedback-based improvements
        - Human preference alignment

Fig 2.

Overview of LLMs with encoder-decoder and decoder-only Transformer architecture for code generation

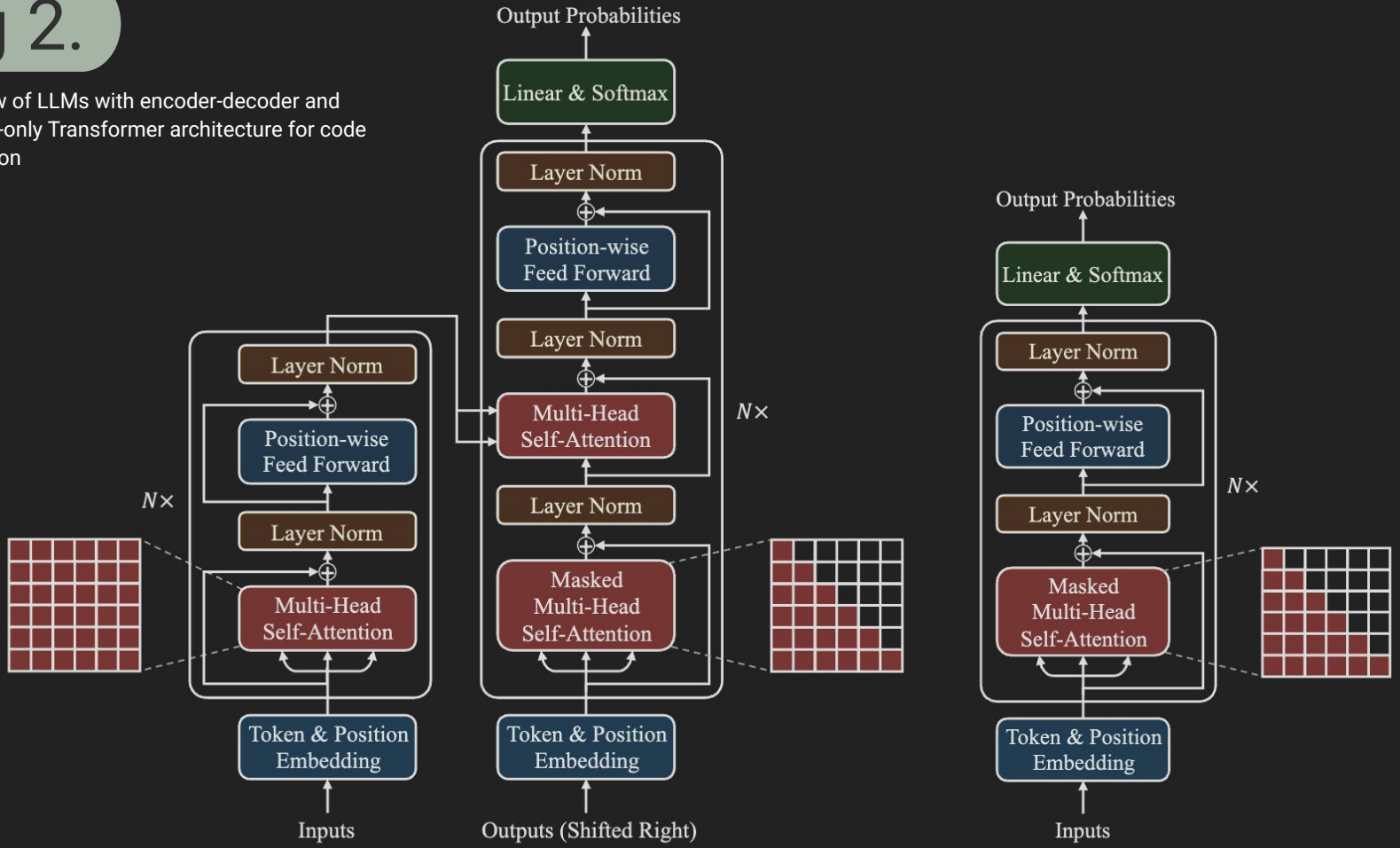


Fig 5.

A diagram illustrating the general training, inference, and evaluation workflow for Code LLMs and their associated databases

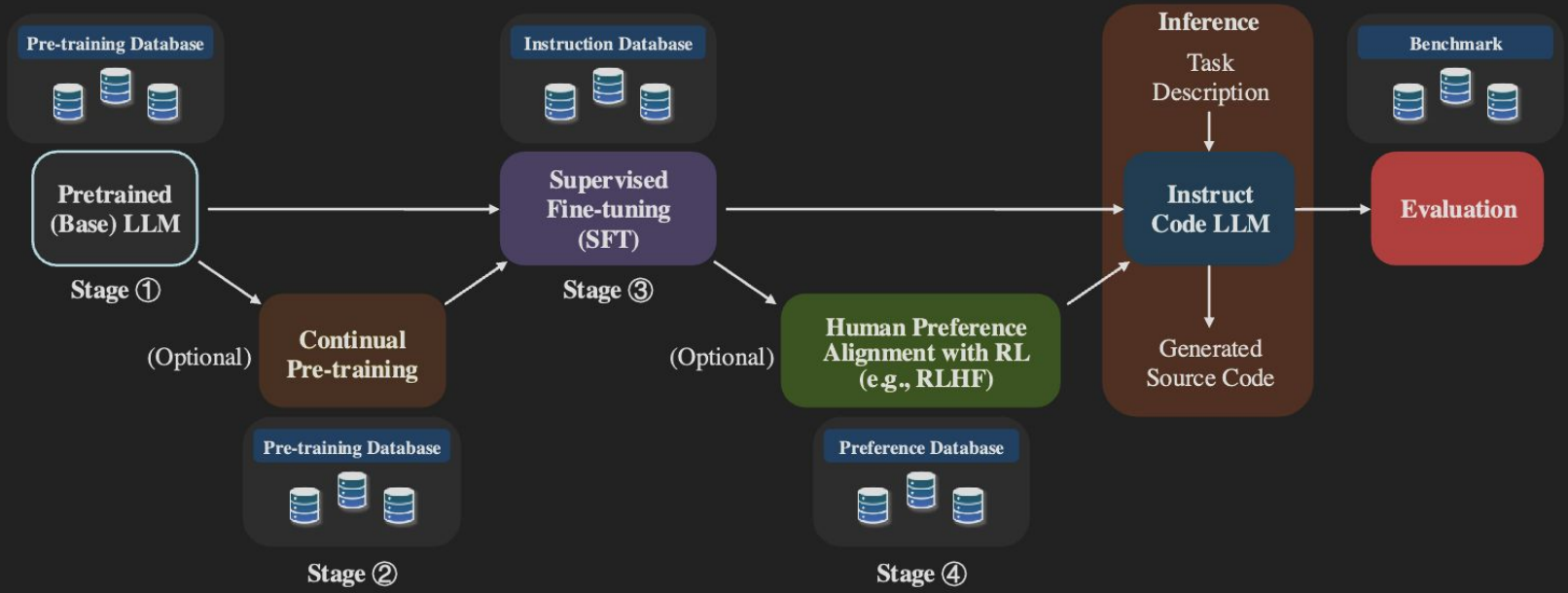
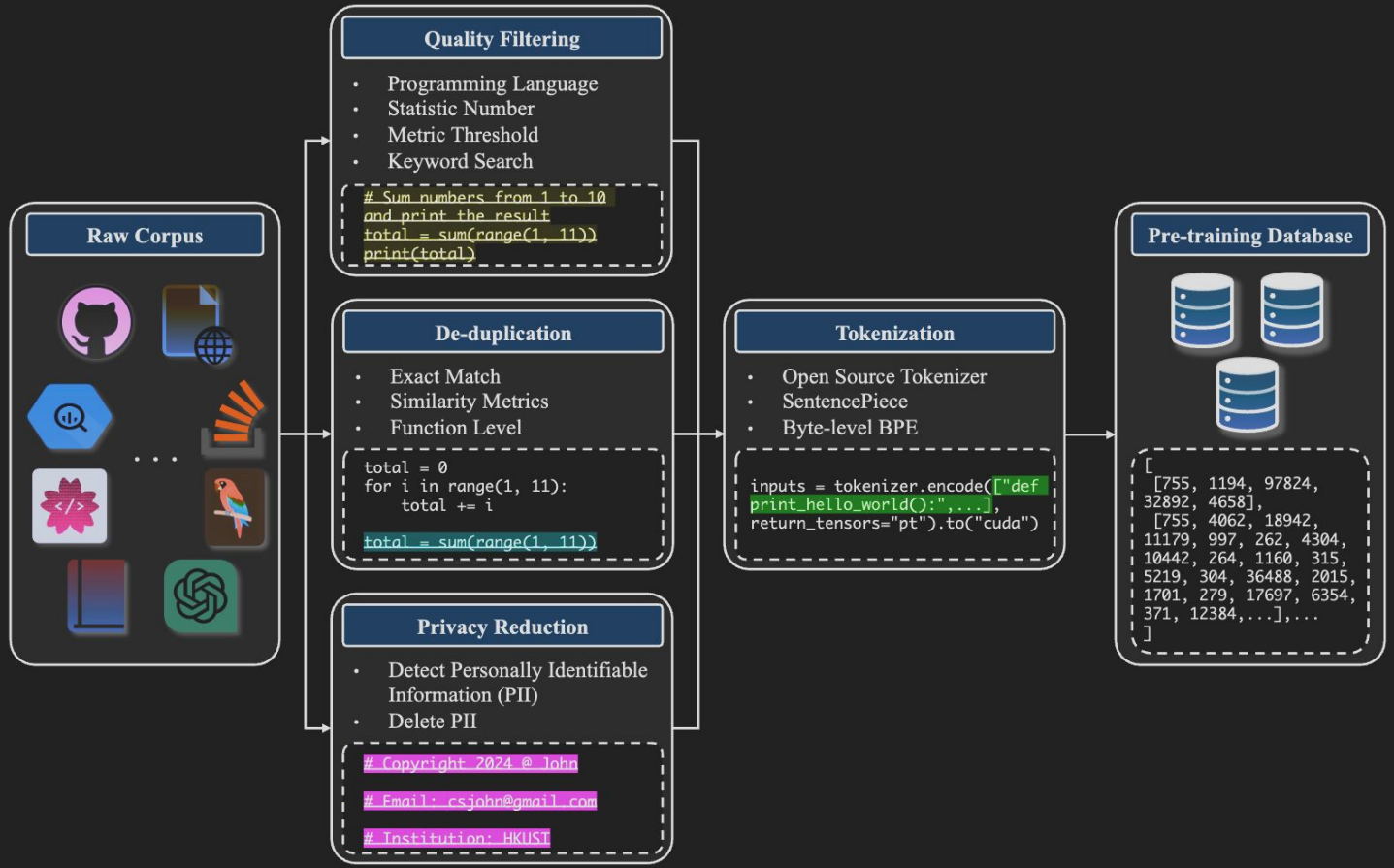


Fig 7.

A diagram depicting the standard data preprocessing workflow utilized in the pre-training phase of LLMs for code generation



Research Questions

RQ1:
How can we **categorize and evaluate** the latest advances in **LLMs for code generation**?

RQ2:
What are the **key insights** into LLMs for code generation?

RQ3:
What are the **critical challenges** and **promising research opportunities** in LLMs for code generation?

Key Contributions

1. **Taxonomy of Code LLMs**
   - Categorizes LLM advancements across data curation, model training, evaluation, and applications

2. **Benchmark & Performance Analysis**
   - Compares LLM performance across HumanEval, MBPP, and BigCodeBench
   - Highlights the performance gap between open-source and closed-source models.

3. **Domain Insights from different communities**
   - Provides of NLP and Software Engineering viewpoints
   - Examines the carbon footprint and computational cost of training LLMs
   - Discusses bias, fairness, and responsible AI practices.

Fig 1.

A chronological overview of large language models (LLMs) for code generation in recent years.

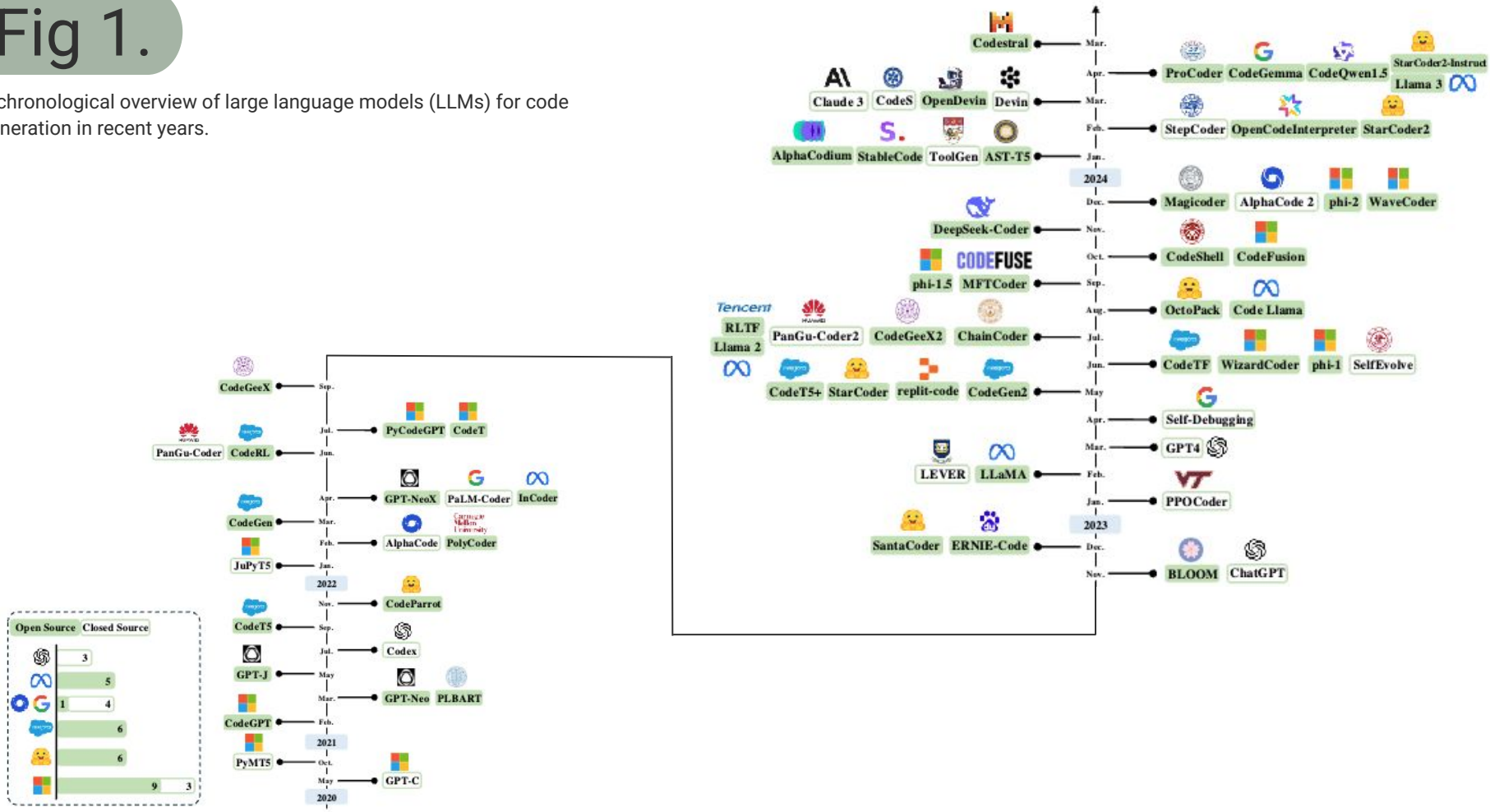


Fig 3.

Overview of the paper search and collection process.

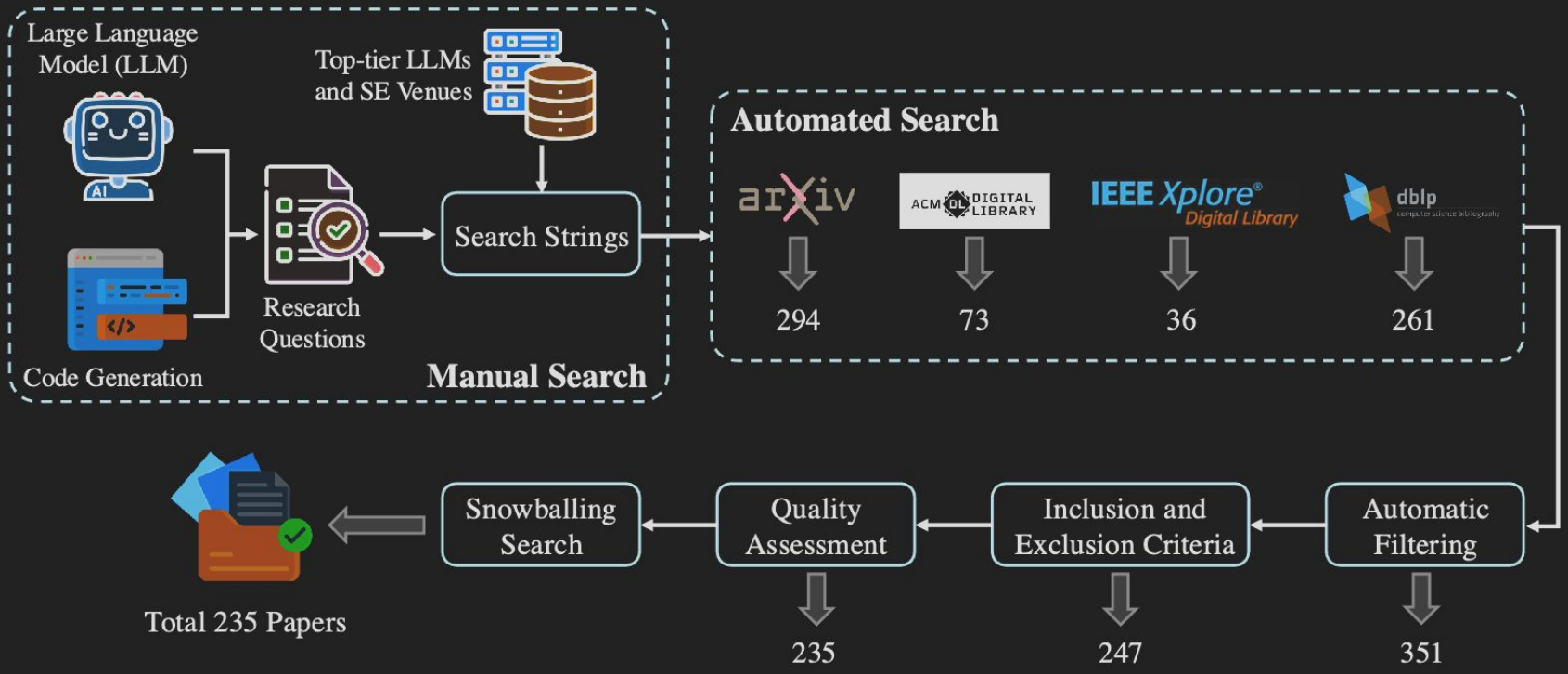
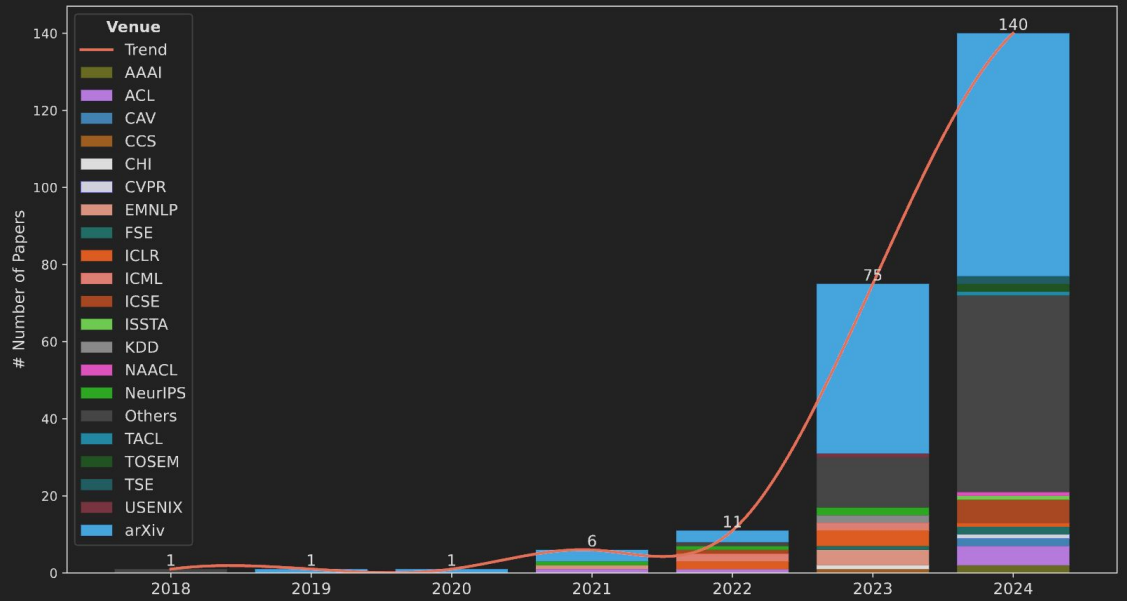
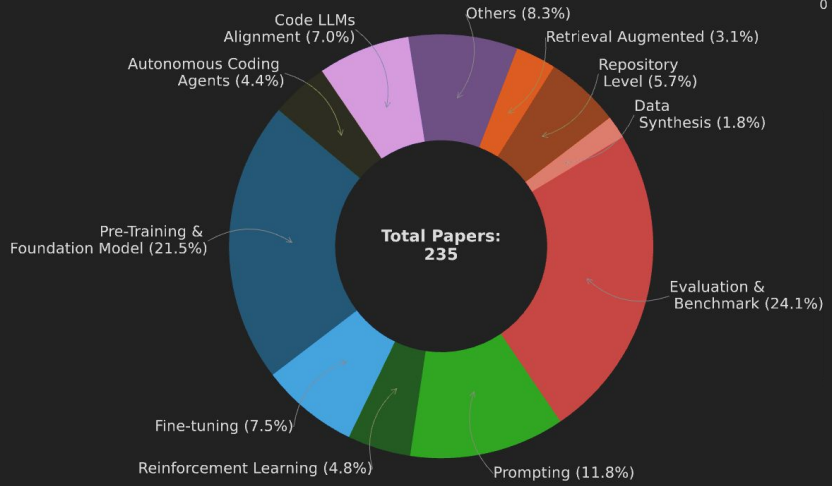


Fig 4.

Data qualitative analysis.
Top: Annual distribution of selected papers across various publication venues.
Bottom: Distribution analysis of research topics covered in the included papers.





Benchmarks

Competition-Based
- **APPS**
(AI Programming Performance System)
- **CodeContests**
(Competitive coding tasks from platforms like Codeforces)

Multilingual
- **HumanEval-X**
(Supports Python, Java, C++, JavaScript, Go)
- **MBXP & xCodeEval**
(Measure performance across multiple programming languages)

General Purpose
- **HumanEval**
(Python, functional correctness)
- **MBPP**
(Entry-level programming problems)
- **BigCodeBench**
(Multi-domain complex tasks)

Data Science-Oriented
- **DS-1000**
(Data science and ML-focused problems)
- **ExeDS**
(Execution-based evaluation of statistical learning models)

Repository-Level
- **RepoEval**
(Codebase comprehension and bug-fixing)
- **SWE-bench**
(Software engineering challenges requiring reasoning and modification)

Models

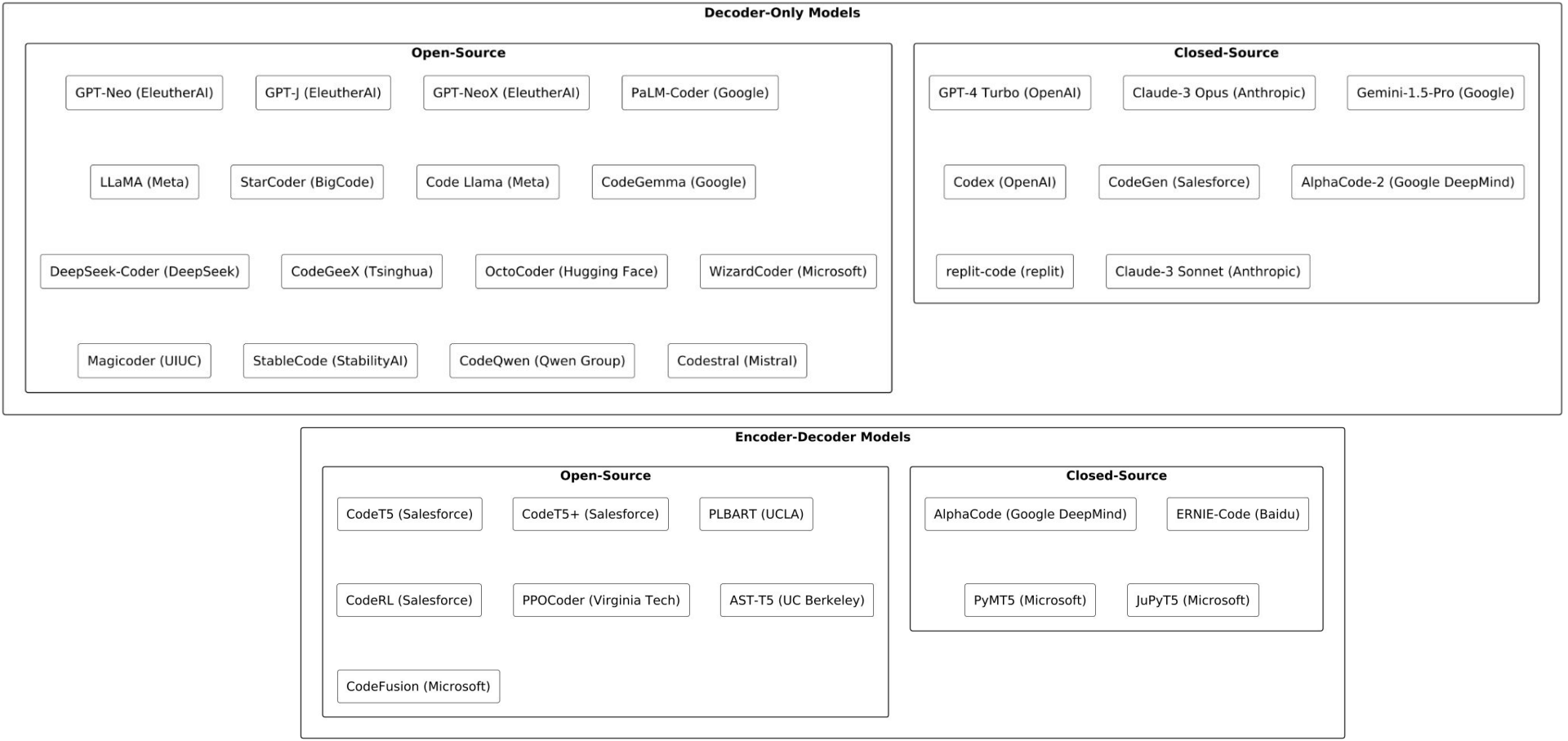
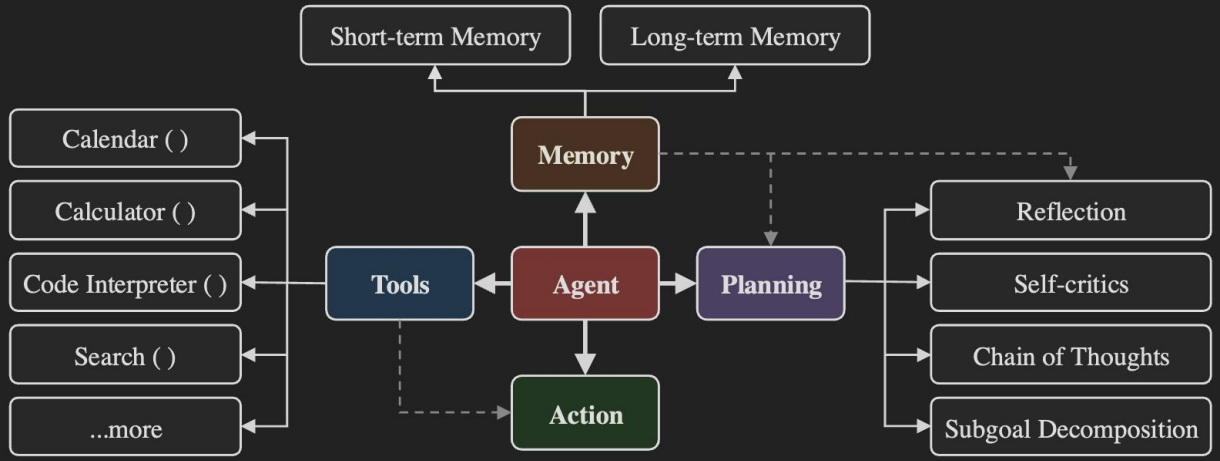

Decoder-Only Models
Open-Source
GPT-Neo (EleutherAI)
GPT-J (EleutherAI)
GPT-NeoX (EleutherAI)
PaLM-Coder (Google)
LLaMA (Meta)
StarCoder (BigCode)
Code Llama (Meta)
CodeGemma (Google)
DeepSeek-Coder (DeepSeek)
CodeGeeX (Tsinghua)
OctoCoder (Hugging Face)
WizardCoder (Microsoft)
Magicoder (UIUC)
StableCode (StabilityAI)
CodeQwen (Qwen Group)
Codestral (Mistral)
Closed-Source
GPT-4 Turbo (OpenAI)
Claude-3 Opus (Anthropic)
Gemini-1.5-Pro (Google)
Codex (OpenAI)
CodeGen (Salesforce)
AlphaCode-2 (Google DeepMind)
replit-code (replit)
Claude-3 Sonnet (Anthropic)
Encoder-Decoder Models
Open-Source
CodeT5 (Salesforce)
CodeT5+ (Salesforce)
PLBART (UCLA)
CodeRL (Salesforce)
PPOCoder (Virginia Tech)
AST-T5 (UC Berkeley)
CodeFusion (Microsoft)
Closed-Source
AlphaCode (Google DeepMind)
ERNIE-Code (Baidu)
PyMT5 (Microsoft)
JuPyT5 (Microsoft)

Fig 14.

The general architecture of an LLM-powered autonomous agent system



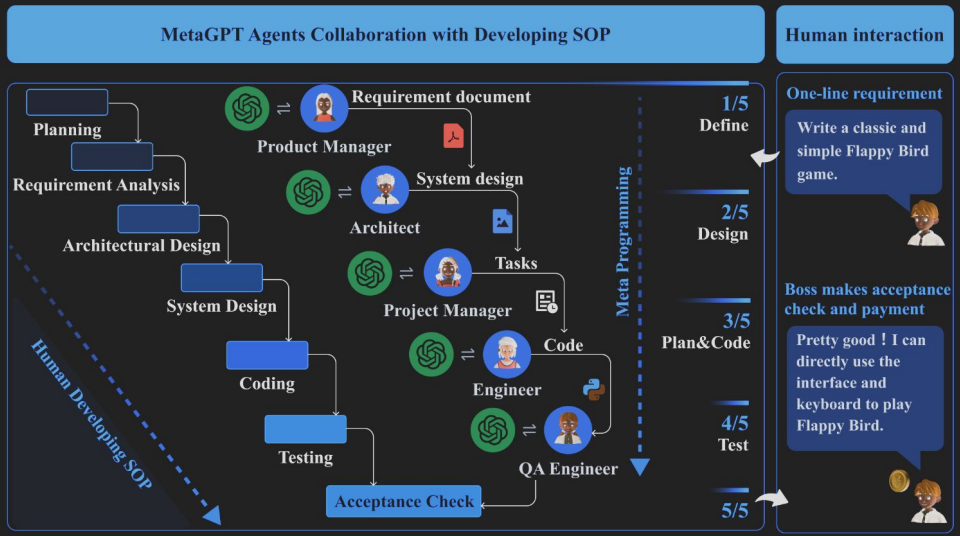
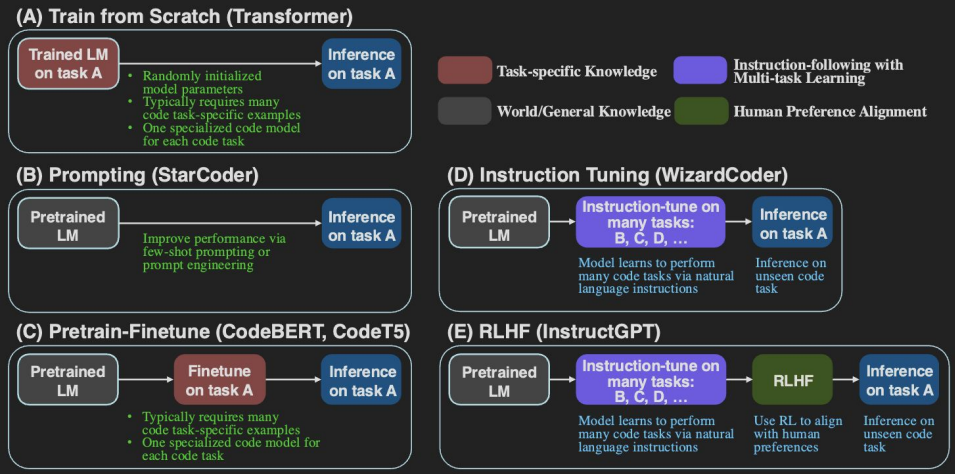


Fig 15.

MetaGPT integrates human workflow efficiencies into LLM-based multi-agent collaboration to break
down complex code-related tasks into specific, actionable procedures.

Fig 9.

Comparison of instruction tuning with various fine-tuning strategies and prompting for code tasks



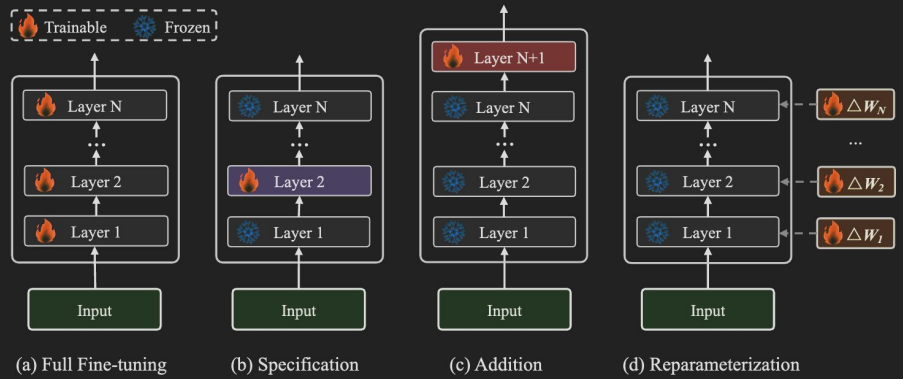


Fig 10.

An illustration of full parameter fine-tuning (FFT) and parameter-efficient fine-tuning (PEFT) methods.

Fig 13.

A workflow illustration of the Retrieval-Augmented Code Generation (RACG).

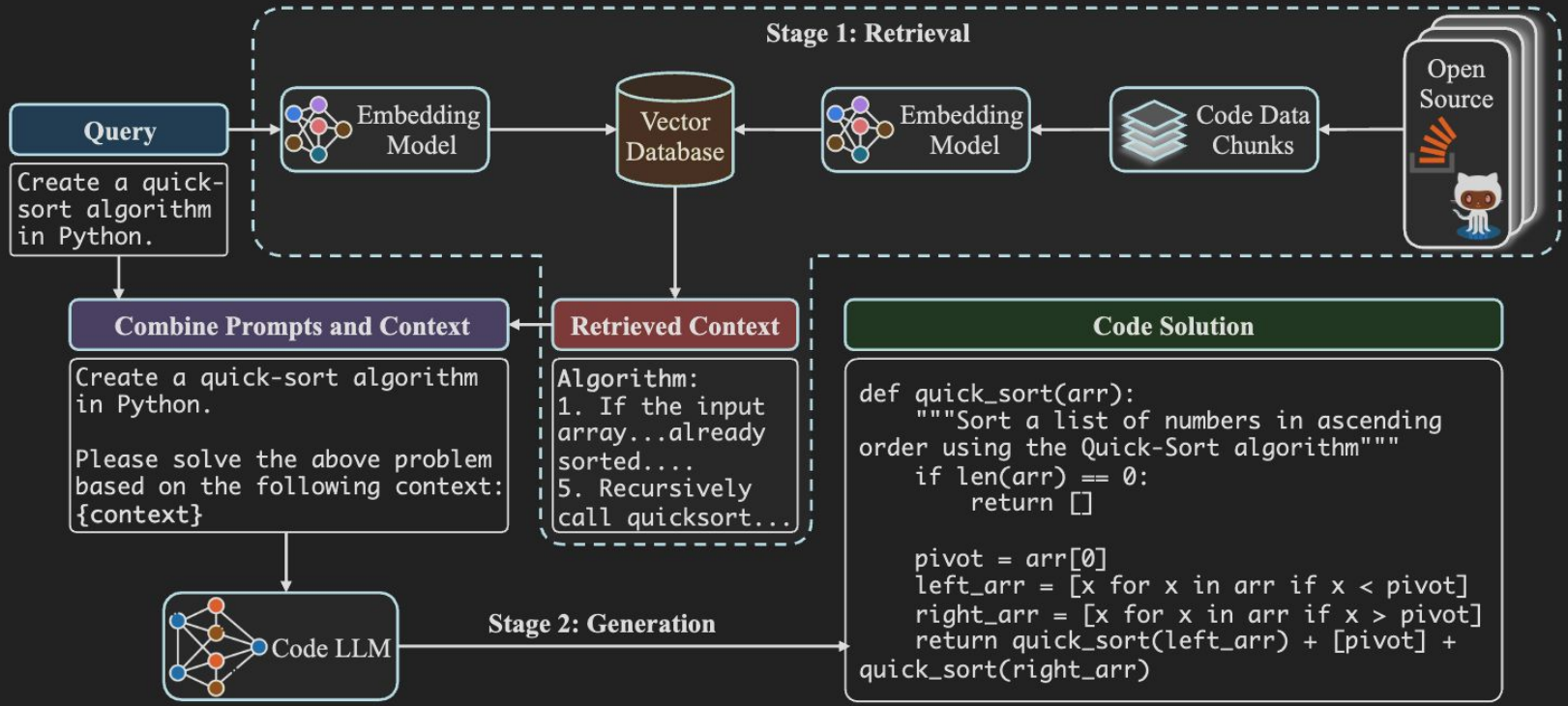
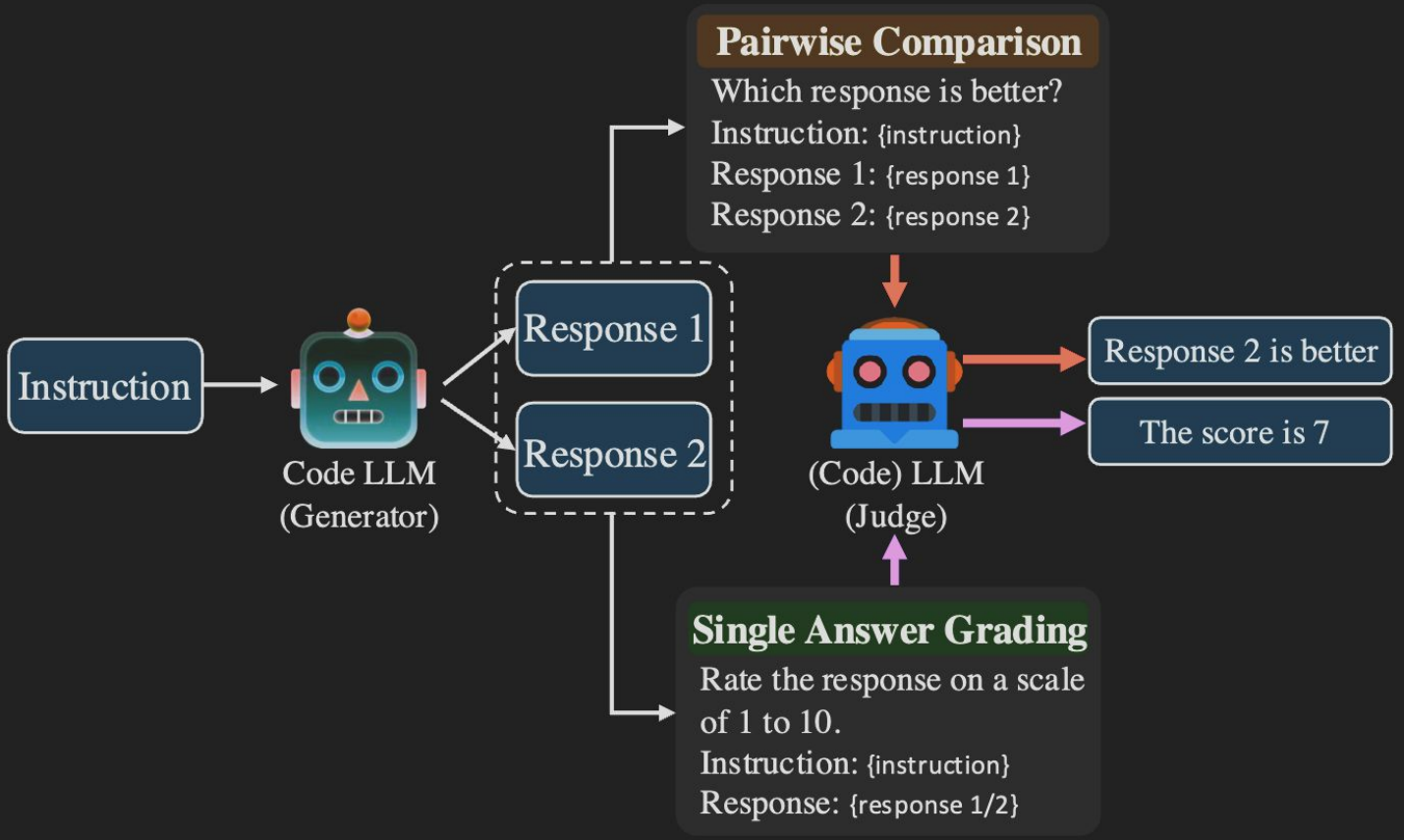


Fig 16.

The pipeline of (Code) LLM-as-a-judge for evaluating generated code by Code LLMs.

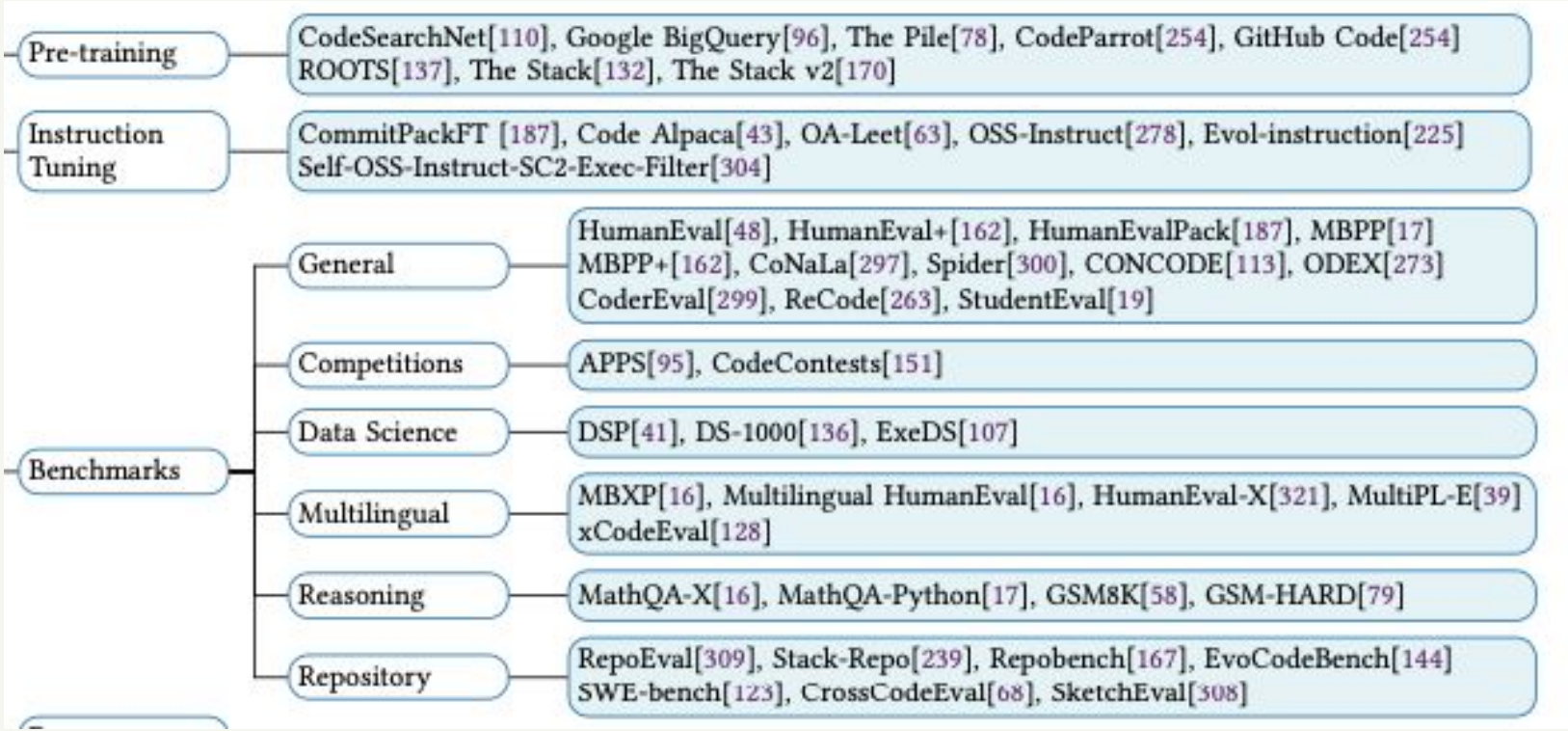


Assumptions & Limitations

1. **Scope, Benchmark & Model Evaluation:**
   - HumanEval, MBPP, and Big CodeBench assumed to be representative of real-world programming challenges.
   - Factors like maintainability, security, and runtime efficiency not considered
   - LLM-as-a-Judge methods introduce biases
2. **Fast Advancement of AI:**
   - Survey captures research up to early 2024, major advancements have taken place
3. **Database & Evaluation Bias:**
   - Study relies on publicly available datasets
   - Bias towards papers from leading conferences and research labs.
   - Most benchmarks focus on Python-based code generation
4. **Computational & Environmental Costs:**
   - LLM training requires substantial energy, contributing to high carbon emissions
   - Quantization and knowledge distillation help reduce compute costs but may impact performance
5. **Ethical Constraints:**
   - Bias in training data can perpetuate security vulnerabilities, copyright risks, and exclusionary practices in generated code
   - LLMs could be misused to generate malicious code or automate harmful software development

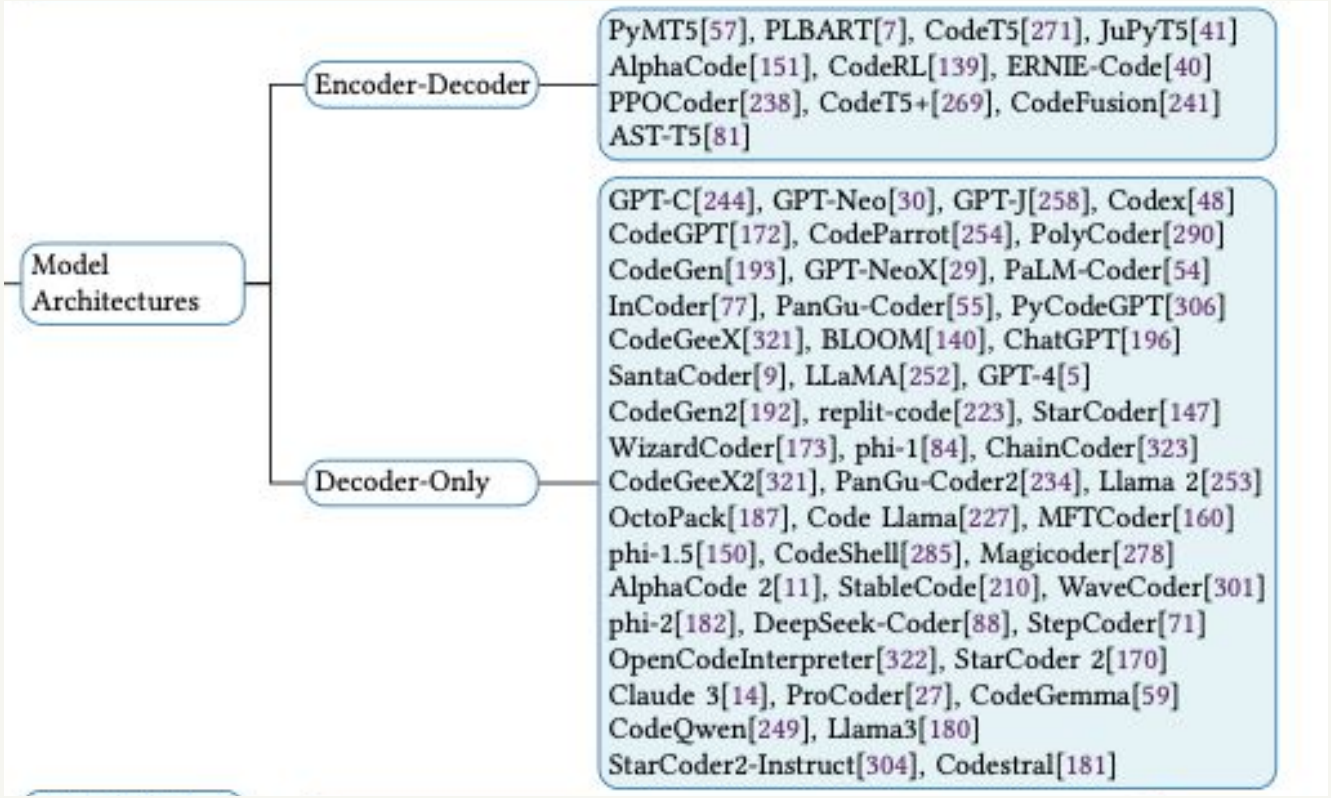
Result (datasets)

Fig 6.



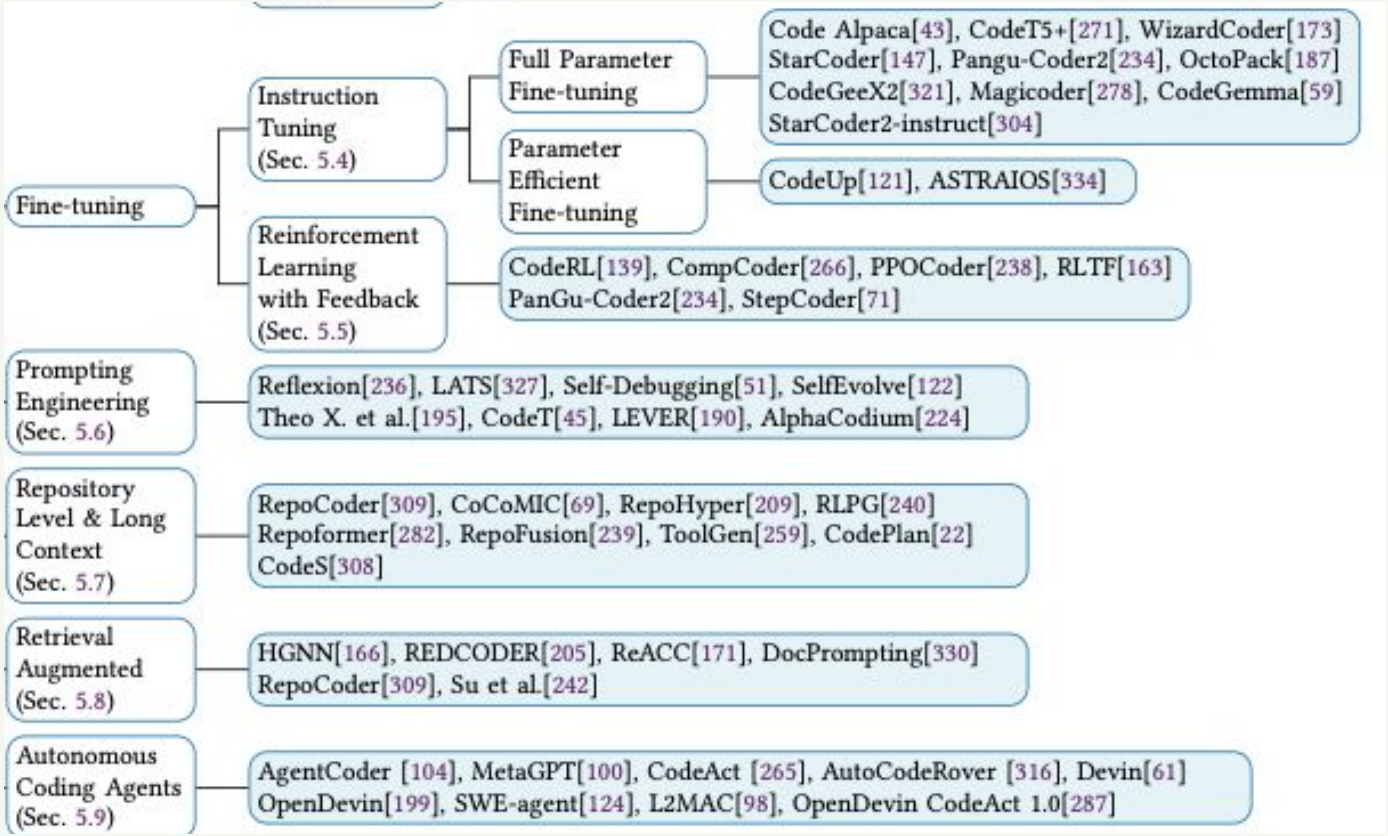
Result (models)

Fig 6.



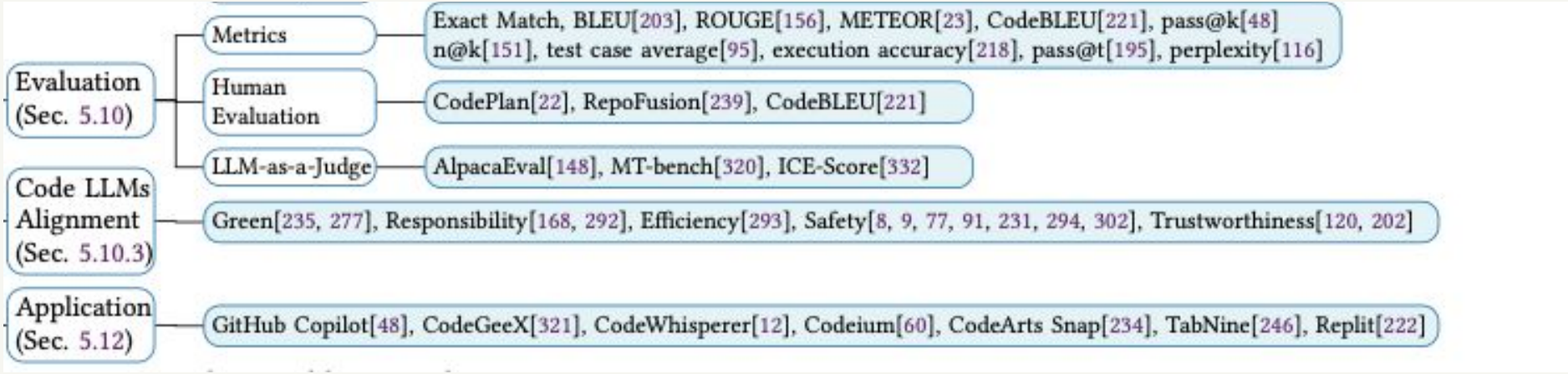
Result (improvement)

Fig 6.



Result (evaluation)

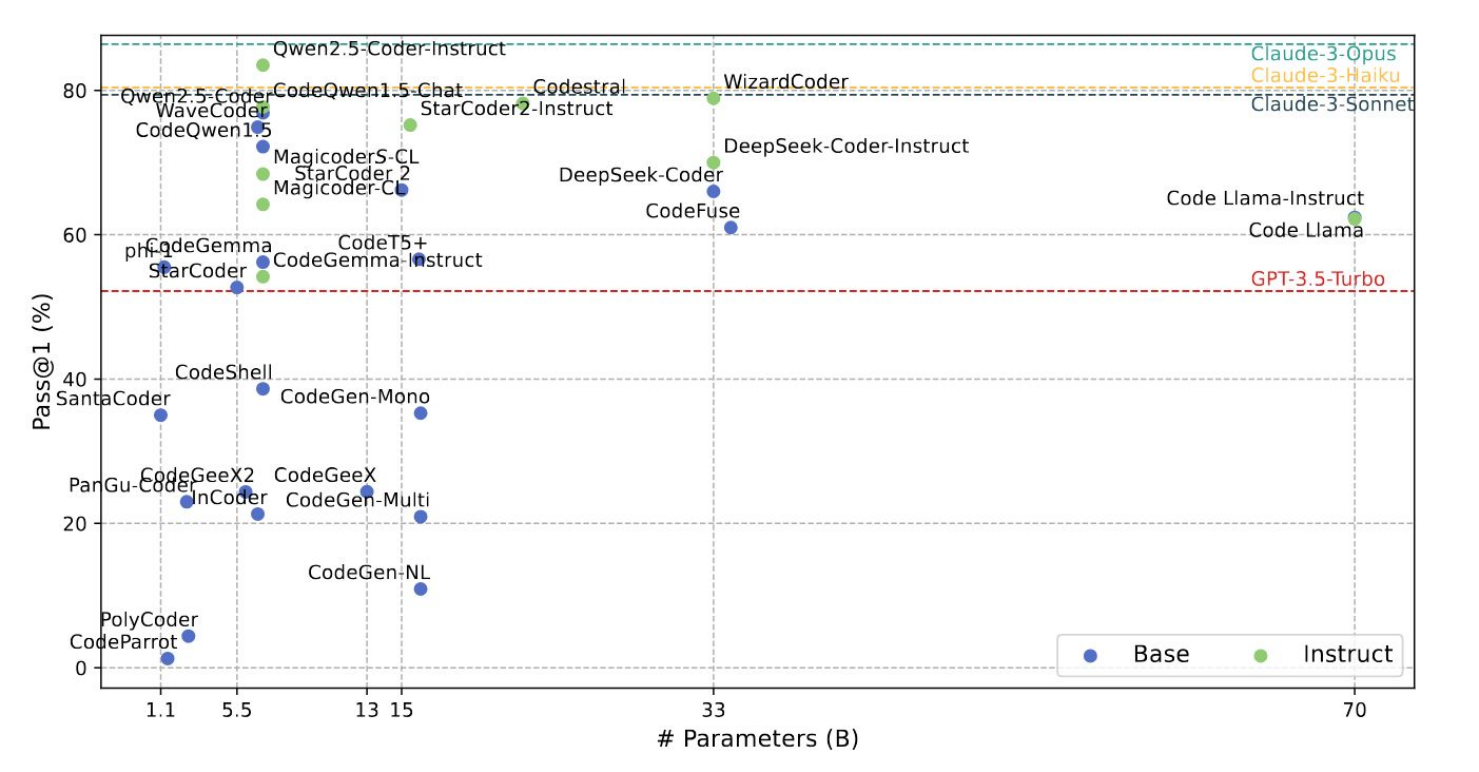
Fig 6.



Result (benchmark)

Fig 17.

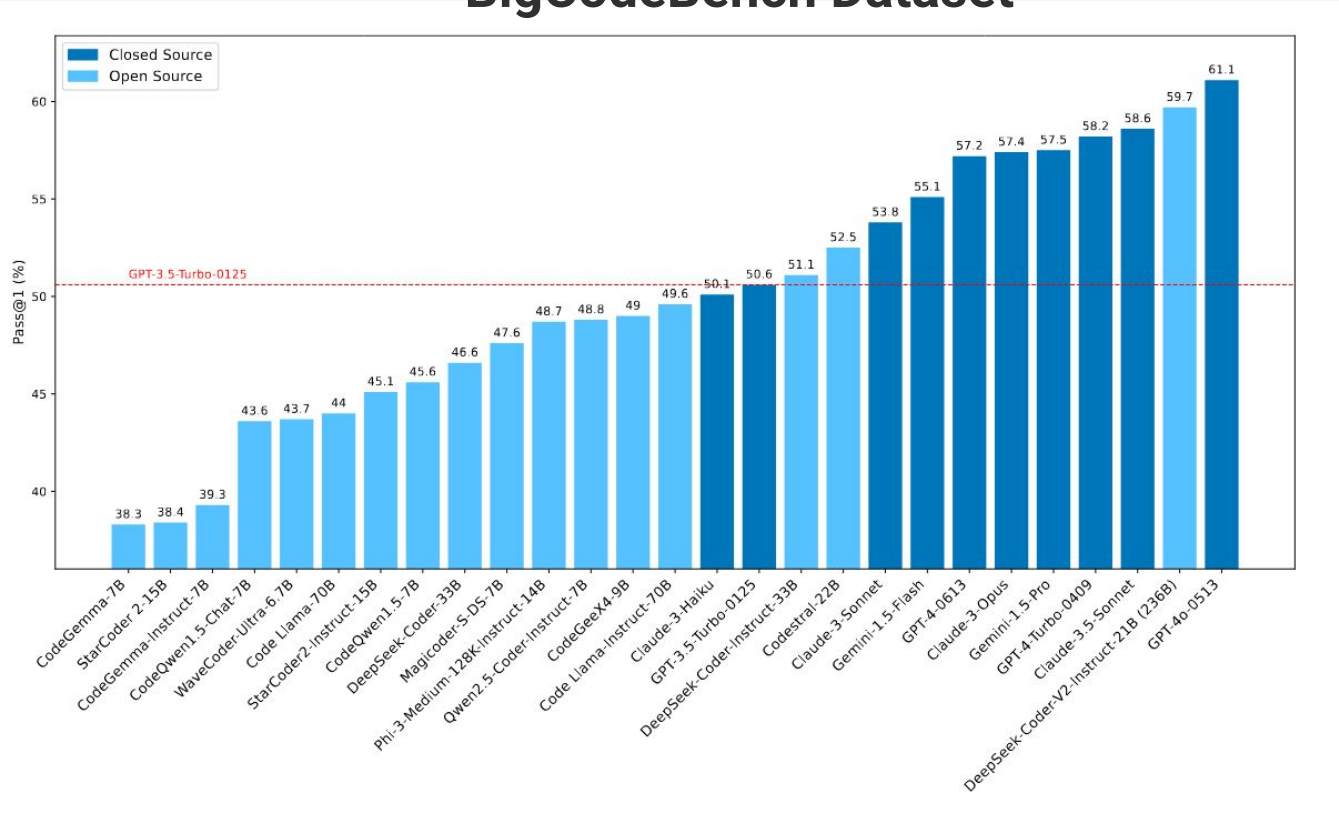
MBPP Dataset



pass@1

Result (benchmark)

Fig 18.



pass@1

Discussions

1. **LLM-alignment**
   - **Green, Responsibility, Efficiency, Safety, and Trustworthiness**
   - Reinforcement Learning with Human Feedback (RLHF)

2. **Innovating model architectures tuned to code structures**
   - Non-transformer-based LLM for code generation?

3. **Continuous learning for LLMs to keep pace with evolving coding knowledge**

Conclusion

This paper presents:

- Systematic literature review in LLMs for code generation.
- Historical overview of evolution of LLMs for code generation
- Empirical comparison of LLMs for code generation with HumanEval, MBPP, and BigCodeBench benchmark.

The authors suggest:

- Future investigation regarding the gap between academia and practical development.

The authors optimistically believe that LLM will ultimately change all aspects of coding and automatically write safe, helpful, accurate, trustworthy, and controllable code, like professional programmers, and even solve coding problems that currently cannot be solved by humans.


IP & Copyright Abuse
“Prompt Engineers”
Energy Consumption
Malicious Code
THIS IS FINE.

Thank You